ARL-TR-7411 ● SEP 2015

US Army Research Laboratory

High-Bandwidth Tactical-Network Data Analysis in a High-Performance-Computing (HPC) Environment: Transport Protocol (Transmission Control Protocol/User Datagram Protocol [TCP/UDP]) Analysis

by Kenneth D Renard and James R Adametz

ARL-TR-7411 ● SEP 2015

US Army Research Laboratory

High-Bandwidth Tactical-Network Data Analysis in a High-Performance-Computing (HPC) Environment: Transport Protocol (Transmission Control Protocol/User Datagram Protocol [TCP/UDP]) Analysis

by Kenneth D Renard
Computational and Information Sciences Directorate, ARL

James R Adametz
QED Systems, LLC, Aberdeen, MD

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE ***(DD-MM-YYYY)*** September 2015	**2. REPORT TYPE** Final	**3. DATES COVERED (From - To)** 1 July 2012–31 December 2014

4. TITLE AND SUBTITLE High-Bandwidth Tactical-Network Data Analysis in a High-Performance-Computing (HPC) Environment: Transport Protocol (Transmission Control Protocol/User Datagram Protocol [TCP/UDP]) Analysis	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Kenneth D Renard and James R Adametz	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) US Army Research Laboratory ATTN: RDRL-CIH-C Aberdeen Proving Ground, MD 21005-5067	**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TR-7411
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This report describes the measurement, reduction, and basic analysis of transport layer protocol (transmission control protocol and user datagram protocol) data collected from large-scale tactical radio network tests. The large volume of data collected necessitates using distributed processing on high-performance-computing (HPC) clusters to reduce and analyze data in a timely fashion. Traditional methods using database ingest and queries on high-end servers cannot handle the increasing scale of current datasets, and processing times are estimated to be on the order of days versus hours using HPC. The requirement for the use of HPC was anticipated, and efforts to develop this capability resulted in a highly capable process that is expected to handle future large-scale tactical network testing.

15. SUBJECT TERMS
tactical networks, data reduction, high-performance computing, data analysis, big data

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Kenneth D Renard
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	UU	22	**19b. TELEPHONE NUMBER (Include area code)** 410-278-4678

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

List of Tables

1. Introduction

The Army's tactical radio networks carry critical data to command, control, communications, and computers (C4) applications that deliver situational awareness and other essential information to Warfighters. These networks operate in challenged environments, without local infrastructure, and are often highly constrained by size, weight, and power limitations. The applications that use these networks must be able to handle lower bandwidths, higher packet drop rates, and a lower overall quality of service compared with their enterprise counterparts. The communication protocols that carry application traffic are typically designed for higher quality network layers and therefore must be monitored and tuned to achieve optimal performance and reliability. It is essential to evaluate the effectiveness of the overall communications capability as new applications, network devices, and protocols are developed and deployed.

The US Army Test and Evaluation Command conducts large-scale test events, such as the Network Integration Evaluation, multiple times per year where tactical radio networks are configured and deployed on experimental missions. Instrumentation on network nodes continually collects test data, which includes packet capture, device status via Simple Network Management Protocol,[1] GPS position data, and precise timing information. Data from this instrumentation are harvested and gathered to conduct reduction and analysis of various performance metrics. An important distinction of this test versus a typical enterprise network performance test is that data are collected from a select subset of all network nodes, and the aggregate of all collected information is necessary to produce the required analysis. Since dropped packets, packet latency, and jitter are important factors in the performance of the network applications, it is critical to have accurately time-stamped packet observations at all network nodes using a global time synchronization source.[2] Test events generate large amounts of data that standard computing platforms (e.g., high-end servers) cannot process fast enough to support the limited time frame for analysts to produce reports. The need for high-performance computing (HPC) to solve this problem has been well established and documented in a US Army Research Laboratory report on data marshalling.[3]

2. Motivation and Desired Data Products

A large class of network applications (e.g., email, web) requires "reliable" transport of data across a network. Reliability is defined as guaranteed and ordered delivery of data from application to application. Internet Protocol (IP) does not guarantee either of these capabilities, especially in a challenged tactical radio network environment. Use of the Transmission Control Protocol (TCP) transport protocol is the standard method of providing reliability to networked applications. Network applications that do not require these reliability services typically use User Datagram Protocol (UDP) for transport. UDP is handled by the capability presented in this report, but due to its simplicity in this scope, the focus is on TCP.

TCP uses acknowledgment messages, sequence numbering of data bytes, and data caching to achieve reliability and ordering. If IP packets are dropped, a TCP implementation will resend application data that it has cached without any action on the part of the application code. This caching consumes valuable memory resources on both the transmitting and receiving network nodes and must be carefully optimized based on the performance of the IP layer. Using memory and available bandwidth efficiently is the key factor in TCP's performance. TCP performs best in an environment with a low round-trip time and a low drop rate.[4] Neither of these is typical in a tactical radio environment.

To evaluate the efficiency of C4 applications that require reliable transport services, careful examination of TCP protocol performance is necessary. Data collected during test events includes IP packet observations, which are correlated and matched prior to TCP analysis. The CommsIP table[5] is generated from raw data and contains a summary of all IP packets that traverse the tactical network including their sending and receiving times at specific observation points. This table is critical to TCP analysis and its summary format serves to reduce the amount of data read in for analysis instead of reading and parsing entire packets. There are 2 types of data products generated for TCP analysis: a CommsTcpSession table and a CommsTcpFlow table. The layout and definitions in these tables were designed by the Test and Evaluation community.[6]

3. TCP Sessions

The CommsTcpSession table (Table 1) presents the TCP session as it is observed at a single point on the network. Therefore, a single TCP session will be represented in this table for each observation point along the path between end points. While there are typically 2 observation points (one at the "initiator" side and one at the "peer" side), there are instances where packets transit one or more intermediate observation points, thus resulting in one or 2 extra TCP session table entries per additional observation point. The information collected in the CommsTcpSession table identifies the endpoints of the session (IP address and TCP port number of initiator and peer) and byte counts of the data transferred between endpoints.

Table 1 CommsTcpSession table columns

Column Name	Description
cts_id	Integer index of row
cts_deid	Device (instrument) Identifier where observation took place
cts_collpt	Collection point or logical observation point on the network
Mac[8]	Medium Access Control (Mac) (Ethernet) address observed as destination for outgoing packets
subsessionid[8]	Zero-based index of session identifier if endpoints are re-used at a later time
cts_iip	IP Address of TCP session initiator
cts_iport	TCP port number of TCP session initiator
cts_iinitseqnum	Initial sequence number of TCP session initiator
cts_pip	IP Address of the TCP session “peer” (listener, or server)
cts_pport	TCP port of TCP session peer
cts_pinitseqnum	Initial sequence number of TCP session peer
cts_dscp	Differentiated Services Code Point from IPv4 Type-of-Service field
cts_establishedstate	Enumeration of the state of session establishment (e.g., only initial synchronize (SYN) observed, only SYN and synchronize and acknowledge (SYN-ACK) observed, full 3-way handshake observed, session observed in already open state)
cts_establishtime	Time to complete 3-way handshake
cts_endstate	Enumeration of session end state (e.g. FIN or RST flags seen from initiator and/or peer)
cts_sessionstarttime	Absolute time of first packet observed for this session
cts_sessionendtime	Absolute time of last packet observed for this session
cts_appbytesitop	Total number of application bytes sent (TCP payload data) from initiator to peer. This does not include retransmissions.
cts_appbytesptoi	Total number of application bytes sent (TCP payload data) from peer to initiator. This does not include retransmissions.
cts_actualbytesitop	Total number of application bytes sent (TCP payload data) from initiator to peer. This includes application data sent in retransmissions.
cts_actualbytes_ptoi	Total number of application bytes sent (TCP payload data) from peer to initiator. This includes application data sent in retransmissions.
cts_totalbytesitop	Total number of IP payload bytes from initiator to peer. This includes TCP headers and options from all packets including retransmissions.
cts_totalbytesptoi	Total number of IP payload bytes from peer to initiator. This includes TCP headers and options from all packets including retransmissions.
cts_outeripsrc	If packet is tunneled (e.g. GRE), this is the source IP address in the outer IP packet header. It is determined once per session and represents the first observation of a tunneled packet.
cts_outeripdst	If packet is tunneled, this is the destination IP address in the outer IP packet header. It is determined once per session and represents the first observation of a tunneled packet.
cts_qalimiteduseid	For quality assurance purposes, this entry may not be suitable for all types of analysis. Possibly due to known instrumentation malfunctions during this session or other network issues.
cts_qareasoncode	A code specifying the reason for any limited use of this entry. Only valid if LimitedUse field is set.
cts_daglimiteduseid	Reserved for DAG (Data Authentication Group) use
cts_dagreasoncode	Reserved for DAG use
numpktsinit[8]	Number of IP packets sent from TCP Initiator address for this session.
numpktsdst[8]	Number of IP packets sent from TCP Peer address for this session.

There are 3 different types of byte counts in the table per session. AppBytes represents the amount of application layer data sent or received by the application, which is the byte count of the payload from the TCP packet. This does not include any TCP headers or payload data that was retransmitted due to packet drops. ActualBytes is the actual count of TCP payload bytes including re-transmissions. The ratio of AppBytes and ActualBytes gives a measure of efficiency with respect to dropped packets on the network. The third byte count adds in all the overhead of the TCP protocol itself, counting all the bytes of the TCP header, TCP options, and retransmissions. It is effectively the byte count of the IP payloads for the TCP session. Each of these byte counts are recorded for each direction: Initiator-to-Peer and Peer-to-Initiator.

4. TCP Flows

The CommsTcpFlow table (Table 2) focuses on Speed of Service (SoS), or latency, of application data from sender to receiver. This table lists the latency for each segment of a TCP session recorded in the CommsTcpSession table. Entries in the CommsTcpFlow table reference CommsTcpSession table records by their unique identifiers. Each entry in the table includes a starting index and length (in bytes) within the overall sequence number space of the session and the total time it took for that segment to get from one observation point to the next. If a packet is dropped in transit, this time should represent the time the data bytes were eventually received (receipt of retransmission) minus the time it was first transmitted. The number of application bytes represented by each row is arbitrary and depends solely on how many bytes have the same SoS.

Table 2 CommsTcpFlows table columns

Column Name	Description
ctf_id	Index into table
ctf_ctsid	Reference to CommsTcpSession table entry observation at transmitting endpoint
ctf_seqnum	The sequence number in the TCP session for the starting byte of this segment
ctf_chunksize	The number of application-layer bytes in this segment having the same SoS
ctf_senddate	Time of first observation of these application-layer bytes being transmitted
ctf_latency	The SoS for this segment of TCP data
ctf_source	‘I’ or ‘P’: Representing whether the transmitter was the TCP initiator or peer

5. UDP

Applications that do not require reliable delivery of application data typically use UDP (e.g., Voice over IP audio traffic, certain variable message format messages). The reporting of UDP packets is done with 2 tables: CommsUdpUnicast and CommsUdpMulticast. There is no processing or translation necessary between the data source (CommsIP) and these output tables. Creation of these tables is a trivial copy after sorting by multicast or unicast destination address. The rest of this report will focus solely on TCP processing.

6. Identification and Partitioning of TCP Packets for Processing

The TCP processing module gets all of its input from the CommsIP table. The entries in the CommsIP table are based on matching observations of an IP packet at a transmitting side and a receiving side. Within the scope of the processing framework, it was decided to record some TCP information in the CommsIP table because the CommsIP module already reads the packet data into memory and parses it. Without this step in the CommsIP module, TCP processing would have to re-read the raw data and parse packets again, resulting in significant replication of input/output (I/O) operations. The TCP layer information shown in Table 3 is stored in the CommsIP table for use in the TCP analysis. The TransportID or "txp_id" value is a 13-byte string comprised of the concatenation of the IP header fields identified in Table 4. This value is the primary identifier used to distinguish TCP sessions from each other.

Table 3 Transport columns included in CommsIp tables

Column Name	Description
txp_id	13-byte transport identifier string
txp_hash	Hash of "normalized" transport identifier
txp_xport	TCP/UDP port of packet transmitter
txp_rport	TCP/UDP port of packet receiver
txp_istcp	Boolean: True if packet is TCP, false otherwise
tcp_tcp_seq	The sequence number from the TCP header (undefined if not TCP)
txp_tcp_ack	The acknowledgement number from the TCP header (undefined if not TCP)
txp_datalen	The total size of the TCP/UDP payload
txp_tcp_flags	The TCP flags field (undefined if not TCP)

Table 4 Transport identifier components

IP Header Field Length	Description
1 byte	IP Protocol identifier from IP header
4 bytes	Source IPv4 address of packet
4 bytes	Destination IPv4 address of packet
2 bytes	Source port of packet
2 bytes	Destination port of packet

An important consideration in the processing of large amounts of TCP sessions is that the workload be distributed as evenly as possible over the set of processors used to run the reduction software. Since it is essential to have the timing, size, and sequence information for all packets in the session "for both directions", we cannot simply divide the workload by txp_id values. Therefore, a hash (txp_hash) is computed over a "normalized" version of the TransportID such that packets going in both directions will have the same TransportID and hash (Fig. 1). The normalization process organizes the source and destination IPv4 addresses and ports in the TransportID based on a string comparison of the IPv4 addresses.

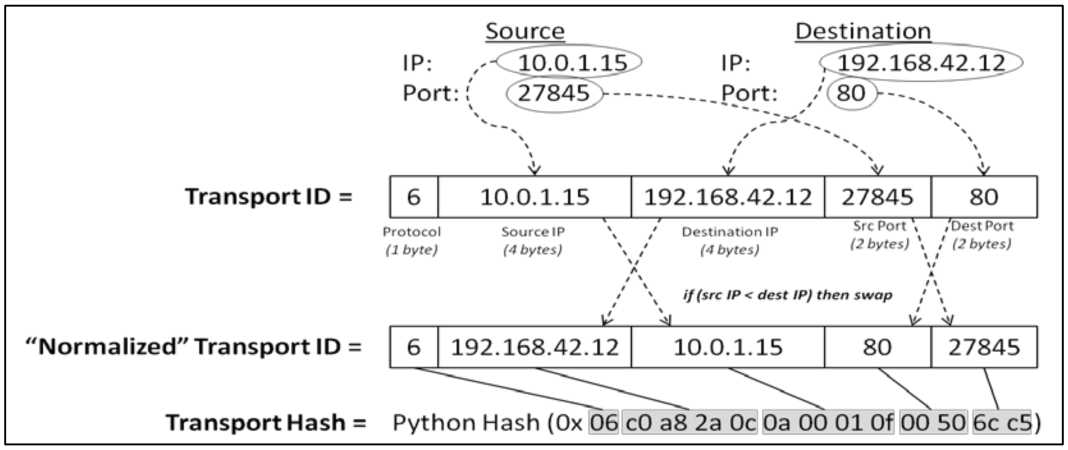

Fig. 1 Generation of transport ID and transport hash

Using just the normalized TransportID for distributing workloads would likely result in an uneven distribution of sessions across processors. This is due to the fact that IP addresses and ports used in most networks are organized into a small set of common subnets and application ports. Instead, a hash of the TransportID is used for distribution of workload. By using the hash of the normalized value, TransportIDs that have only a small difference (e.g., only last octet of source IP is different) will have hashes that are very dissimilar. Assigning processors each a range of txp_hash values to work on results in an even distribution of sessions and keeps both directions of the session together. This approach is a "best effort" at distributing processing load, although may not be optimal. While each processor may get a roughly equal number of TCP sessions to process, the computational workload depends more on the number of packet records to process. TCP sessions with high packet counts upset this balance and are handled with another optimization, which is discussed later.

7. TCP Processing Considerations

7.1 Data Volume

The amount of TCP data in a test can be quite large, and it is only expected to grow in future testing. An HPC compute node has finite memory and no disk space for swap. An out of memory error will crash the node or processes will be killed before the physical memory limit is reached. Writing out intermediate files, which are then read multiple times, can be a significant bottleneck, but random-access memory considerations are the primary factor in how data flows in the TCP processing module. A particular worker process is given a set of TCP sessions to analyze, and it must pick out all relevant packet records from the set of all IP packet records (CommsIP table). This set of TCP packets is written to a temporary file specific to the worker process and used for further processing. It is not safe to load all packet records from all of a worker's TCP sessions at once.

7.2 Incomplete Data

Network data collection devices are subject to errors and congestion just as network devices are. It is understood that any packets of a TCP session may be present but not observed at a particular observation point. A packet dropped by the network may be indistinguishable from a packet dropped by the instrumentation. Therefore, it is not always possible to see the same TCP payload at each end point to compare transmitted bytes and received bytes. Our approach is designed to handle missing packets and give the most complete picture possible of the transport layer performance.

7.3 Port Reuse

A typical run of the TCP analysis code will process packets that were collected over a period of 8 or more hours. It is entirely possible that TCP endpoint pairs (IP and port sources and destinations) will be reused during that time period. Consideration must be given to separate these sessions in time.

7.4 Network Optimization

Since tactical networks often suffer from large round-trip times and packet loss, special hardware might be deployed to minimize the impact these effects have on the performance of TCP. A Performance-Enhancing Proxy (PEP) may be used that effectively "intercepts" a TCP session and relays the application-layer traffic in a more optimal TCP configuration (e.g., larger window sizes). A matched PEP device close to the other endpoint also relays in a similar fashion. When operating with these devices, a single TCP session observed at 2 points on the network will have different initial sequence numbers for the session and TCP segment sizes may have changed en route. It is therefore important to consider "relative sequence numbers" when comparing observations at initiator and peer sides of the session. If the 3-way TCP handshake[8] is not observed at one endpoint (instrumentation was not collecting, or dropped packets), it may be impossible to determine speed of service for the segments of the session.

8. TCP Processing Algorithm

The TCP reduction module is started within the context of the HPC reduction framework.[9] The module depends on output from a completed run of the CommsIP[5] module. Given the set of files containing the CommsIP table, it selects a number of worker tasks to spawn based on the number of message passing interface ranks in the current HPC job, rounded up to the nearest power of 2. Rounding to a power of 2 simplifies the selection of a txp_hash range to bit-mask and compare rather than a range or division operation.[10] Each task is therefore assigned a subset of the txp_hash value space to work on (Fig. 2).

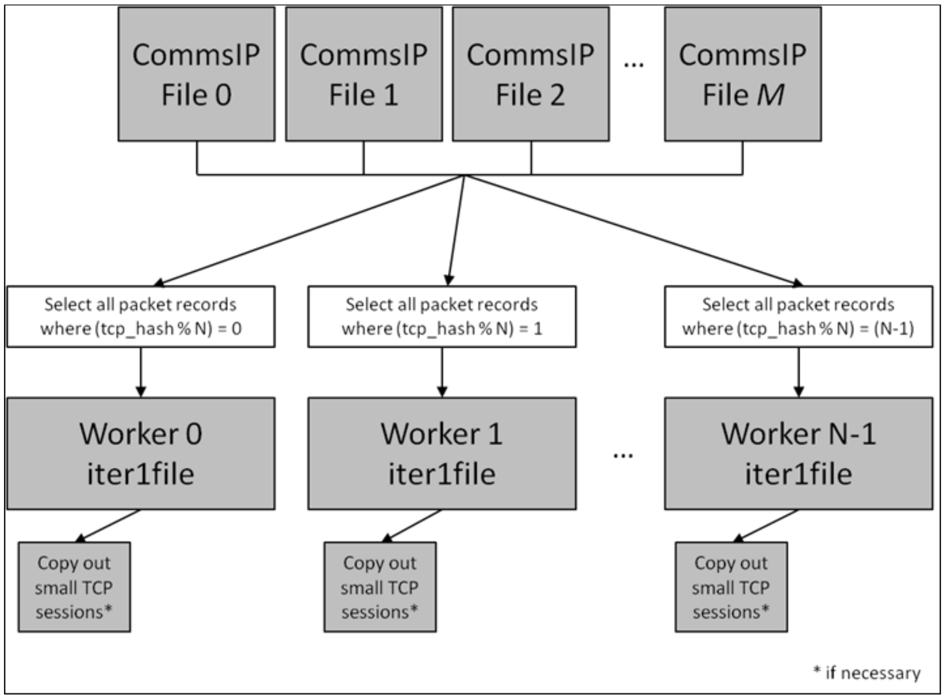
CommsIP File 0
CommsIP File 1
CommsIP File 2
...
CommsIP File M
Select all packet records where (tcp_hash % N) = 0
Select all packet records where (tcp_hash % N) = 1
Select all packet records where (tcp_hash % N) = (N-1)
Worker 0 iter1file
Worker 1 iter1file
...
Worker N-1 iter1file
Copy out small TCP sessions*
Copy out small TCP sessions*
Copy out small TCP sessions*
* if necessary

Fig. 2 TCP module distribution of work

Each task goes through all CommsIP files and selects packet records that are assigned to it. Where N is the number of tasks, the n-th task selects all txp_hash values where the last $log_2(N)$ bits of the txp_hash value are equal to n. It copies these packets into a new file (called “iter1file”) for subsequent processing. While this extra I/O takes time, it is critical to gather all required packets while not exceeding the memory limits of the computing node. This step results in a single file containing all packet records for all sessions that the task will process. Searching this file for packets from each TCP session it processes is much more efficient than searching the entire space of all CommsIP files.

A first pass through the iter1file is made to get a list of all txp_hash values and the TransportID values for each of them. Ideally, there are 2 TransportIDs for each hash. If there is only one, then one direction of the TCP session was not observed. If there are more than 2, a hash collision has occurred and it is handled by matching the TransportIDs themselves. Further processing considers one txp_hash value or a single pair of related TransportIDs at a time.

The iter1file is searched for all TCP SYN and SYN-ACK[8] packets from the pair of TransportIDs. This step results in finding the set TCP establishment handshakes and their associated initial sequence numbers from the session initiator (Fig. 3). This serves to identify the start time of each TCP session that uses the same IP and port endpoints (hereafter called “subsessions”). The results of this step are recorded for each observation point, as sequence numbers may have changed along the path by a PEP. The goal of this step is to discover the number of subsessions and the individual time frames of each subsession so that that all other packets may be grouped and processed appropriately.

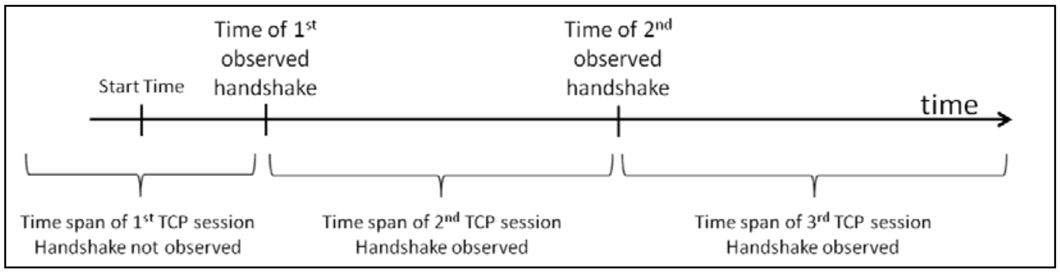
Start Time
Time of 1st observed handshake
Time of 2nd observed handshake
time
Time span of 1st TCP session Handshake not observed
Time span of 2nd TCP session Handshake observed
Time span of 3rd TCP session Handshake observed

Fig. 3 Time ranges for subsessions when TCP ports are reused

Another pass is made through the iter1file, finding all packets with the TransportID being processed. This time, each packet is added to a subsession tracker and is identified as either a transmitted or received packet at a particular observation point. Each packet added updates, counters, status flags, and a list of data segments (sequence number, data length, time) and ACKs (ACK number, time). A single packet record from the iter1file is usually added as both a transmitted packet and a received packet at their respective observation points. Each subsession object tries to identify the packet flow as being the initiator or peer of the TCP session. This is done based on the observation of SYN or SYN-ACK packets in the flow.

The next step is to pair appropriate transmit and receive subsession objects at each observation point. In the ideal case, there is exactly one transmitted subsession and one received subsession and one of these has been identified as the initiator side and the other as the peer side. A CommsTcpSession record is created with the observation point and session statistics. The app_bytes fields (cts_appbytesitop and cts_appbytesptoi) are populated by going through the segment list and ignoring bytes that had previously been sent based on sequence numbers and lengths. If there are not exactly one transmitted and one received subsessions, then initial sequence numbers are used in an attempt to match appropriate observed flows. Since this considers only one observation point, the effects of a PEP and nonmatching sequence numbers are not an issue.

The SoS calculation is made by recording the time of first transmission of a segment (sequence number and data length) and the time of first acknowledgment of that segment (Fig. 4). Previous approaches reconstructed the payload data into 750-byte “chunks” and compared the time sent versus received. A per-segment approach gives a more granular result and handles unobserved packets by either showing no SoS, or representing a slightly longer SoS value. For example, if 2 TCP segments are transmitted representing 100 bytes each, there may have been a single ACK representing all 200 bytes, or 2 separate ACKs, representing each segment independently. These 2 scenarios are indistinguishable if the first independent ACK is not recorded.

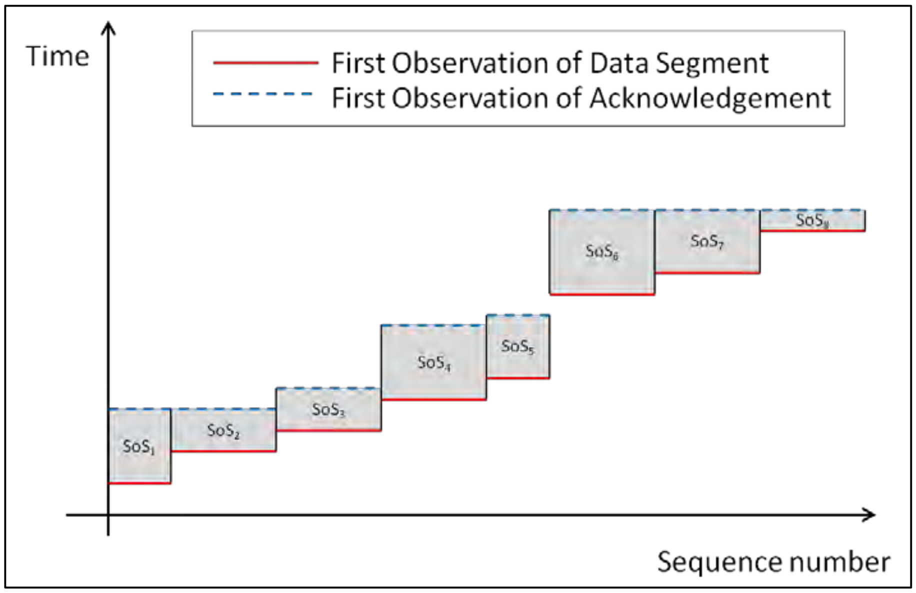

Fig. 4 Using transmitted data segments and transmitted ACKs to determine SoS of TCP payload data

All trackers for tramsmitted subsessions, across all observation points, that were identified are gathered and paired for the SoS analysis. The first grouping is done temporally, looking for time overlap. Then initiator and peer subsessions are separated such that they can be paired together. If there are more than one set of initiators and peers in a subsession, initiators and peers from the same observation point are not paired with each other. This handles the case where packets from point “A” to point “C” are routed through point “B”. This case is recorded as 2 separate speeds of service records between points “A” and “B” and between points “B” and “C”.

Considering a flow of traffic from a TCP initiator to a TCP peer, the transmitted data segments from the initiator’s observation point are compared against the transmitted ACKs from the peer. During the processing of the TCP Session data, the starting sequence number and length from each data segment are recorded. Retransmitted bytes are not considered for this purpose since flow analysis is only concerned with the earliest transmission of data. This is combined with a list of ACKs and their earliest times. For example, if a TCP endpoint sends 3 packets with the same ACK value, only the earliest observation is recorded. The resulting list of data segments and ACKs are used to construct a list of latencies for segments of the application layer data stream. Each data segment and its latency is recorded as an entry in the CommsTcpFlows table.

9. TCP Processing Optimizations

Optimizations have been made to reduce memory requirements and I/O time in processing sessions. In initial prototypes of the code, information from all packet records for a session were loaded into memory and then sorted by both transmit and receive time order into separate lists. This led to long sorting times and memory exhaustion when processing large sessions. Now, a preindexed sorting mechanism allows packet records to be loaded in both transmitted and received time order. Retransmitted data segments and duplicate acknowledgements can then be ignored since only the earliest observation is used.

The TCP sessions assigned to a worker are processed sequentially. During this time, each packet record from the session must be read from the iter1file. As the number of TCP sessions assigned to a worker increases, so does the number of passes through the iter1file. A bottleneck appeared in TCP sessions with extremely large packet counts, resulting in very long read times for ALL sessions handled by that worker. To alleviate this, smaller TCP sessions (ones with fewer packets) are copied out to another file. While this requires another round of I/O, it results in many passes through a smaller file plus a small number of passes through the original large file. In some cases, this optimization resulted in a 100× speedup for workers.

10. Conclusions

The TCP processing module developed for HPC has been used during several test events over the course of one and a half years. It has evolved as requirements change and optimizations are discovered. Large-scale tests can process on the order of one million TCP sessions in one to 3 h, depending mostly on the distribution of packets among the sessions. The types of analysis that are done are unique; they utilize observations from both endpoints of the session and compute speed of service based on the time that data is transmitted from the sender to the time an acknowledgement is sent by the receiver. While this module was designed and built for C4 analysis of test events, it could easily be adapted for enterprise or wide-area network analysis using appropriate data collection tools.

11. References and Notes

1. Simple Network Management Protocol, which is primarily used to query systems under test for status information.

2. Panneton B, Adametz J, Franssen J. High-bandwidth tactical-network data analysis in a high-performance-computing (HPC)_environment: time tagging the data. Aberdeen Proving Ground (MD): Army Research Laboratory (US); 2015 Sep. Report No.: ARL-CR-0778.

3. Renard K, Rivera JD, Adametz J, Franssen J. High-bandwidth tactical-network data analysis in a high-performance-computing (HPC) environment: data marshalling. Aberdeen Proving Ground (MD): Army Research Laboratory (US); 2015 Sep. Report No.: ARL-TR-7410.

4. Huston, G. TCP performance. The Internet Protocol Journal. 2000:3(2):2–24.

5. Panneton B, Adametz J. High-bandwidth tactical network data analysis in a high-performance-computing (HPC) environment: packet-level analysis. Aberdeen Proving Ground (MD): Army Research Laboratory (US); 2015 Sep. Report No.: ARL-CR-0779.

6. Adametz J. Army Test Center, Analysis. C4 data model description document 1.8.13. Aberdeen Proving Ground (MD): Army Research Laboratory (US); Aberdeen Test Center; not yet published.

7. These fields are present in the working data store in the HPC processing modules, but they are not exported to the final C4 Data Model.

8. Postel J. Transmission control protocol. 1981 Sep. [accessed 2014 Jan 1]. http://www.rfc-editor.org/info/rfc793.

9. Panneton B, Adametz J. High-bandwidth tacticalnetwork data analysis in a high-performance-computing (HPC) environment: HPC data reduction framework. Aberdeen Proving Ground (MD): Army Research Laboratory (US); 2015 Sep. Report No.: ARL-CR-0777.

10. Modulo operation chosen to make PyTable “where” statement query more efficient in selecting values.

List of Symbols, Abbreviations, and Acronyms

C4	command, control, communications, and computers
DAG	Data Authentication Group
HPC	high-performance computing
I/O	input/output
IP	Internet Protocol
PEP	Performance-Enhancing Proxy
SoS	Speed of Service
SYN	synchronize
SYN-ACK	synchronize and acknowledge
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
2 (PDF)	DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL IMAL HRA MAIL & RECORDS MGMT
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
1 (PDF)	DIR USARL RDRL CIH C K RENARD

INTENTIONALLY LEFT BLANK.